CSC 446 Notes: Lecture 1

Typed by Darcey Riley

January 24, 2012

1 What Is Machine Learning?

Machine learning is about automatically analyzing data; it mostly focuses on the problems of classification and regression. In this class, we will learn multiple methods for solving each of these problems.

- **Classification** is the problem of assigning datapoints to discrete categories; the goal is to pick the best category for each datapoint.
- **Regression** is the problem of learning a function from datapoints to numbers; fitting a line or a curve to the data is an example of a regression problem.

One example of a classification problem would be: given heights and weights, classify people by sex. We are given a number of training datapoints, whose heights, weights, and sexes we know; we can plot these datapoints in a two-dimensional space. Our goal is to learn a rule from these datapoints that will allow us to classify other people whose heights and weights are known but whose sex is unknown. This rule will take the form of a curve in our two-dimensional space: then, when confronted with future datapoints, we will classify those datapoints which fall below the curve as female those which fall above the curve as male. So how should we draw this curve?

One idea would be to draw a winding curve which carefully separates the datapoints, assuring that all males are on one side and all females are on the other. But this is a very complicated rule, and it's likely to match our training data too closely and not generalize well to new data. Another choice would be to draw a straight line; this is a much simpler rule which is likely to do better on new data, but it does not classify all of the training datapoints correctly. This is an example of a fundamental tradeoff in machine learning, that of **overfitting** vs. **generalization**. We will return to this tradeoff many times during this class, as we learn methods of preventing overfitting.

2 Logistics

Homework comprises 35% of the grade for this class; there will be both written and programming asssignments. The written assignments will be similar to exam problems. The programming assignments, to be done in Matlab, will provide demonstrations of these machine learning algorithms in action. Absolutely no late homework will be accepted. There will be review sessions on the following dates:

- Matlab help session (Darcey): Friday, January 20th, 3PM in the software lab (CSB 727)
- Vector calculus help session (Dan): Monday, January 23rd, 11AM in CSB 601

Matlab is available through the CS department network. Those who do not have accounts on this network should get an account by speaking to Marty Guenther on the seventh floor. Although students are permitted to use Octave (the free version of Matlab) to complete their assignments, the CS department account is still required for turning in the assignments, so everyone should have one regardless.

3 Probability Theory

This section contains a quick review of basic concepts from probability theory.

Let X be a **random variable**, i.e. a variable that can take on various values, each with a certain probability. Let x be one of those values. Then we denote the probability that $X = x$ as $P(X = x)$. (We will often write this less formally, as just $P(x)$, leaving it implicit which random variable we are discussing. We will also use $P(X)$ to refer to the entire probability distribution over possible values of X.)

In order for $P(X)$ to be a valid probability distribution, it must satisfy the following properties:

- For all x, $P(X = x) \geq 0$.
- $\sum_x P(X = x) = 1$ or $\int P(x)\mathrm{d}x = 1$, depending on whether the probability distribution is discrete or continuous.

If we have two random variables, X and Y, we can define the **joint distribution** over X and Y, denoted $P(X = x, Y = y)$. The comma is like a logical "and"; this is the probability that both $X = x$ and $Y = y$. Analogously to the probability distributions for a single random variable, the joint distribution must obey the properties that for all x and for all y, $P(X = x, Y = y) \geq 0$ and either $\sum_{x,y} P(X = x, Y = y) = 1$ or $\int\int P(x, y)\mathrm{d}y\mathrm{d}x = 1$, depending on whether the distribution is discrete or continuous.

From the joint distribution $P(X, Y)$, we can **marginalize** to get the distribution $P(X)$: namely, $P(X = x) = \sum_y P(X = x, Y = y)$. We can also define the **conditional probability** $P(X = x|Y = y)$, the probability that $X = x$ given that we already know $Y = y$. This is $P(X = x|Y = y) = \frac{P(X=x,Y=y)}{P(Y=y)}$, which is known as the **product rule**. Through two applications of the product rule, we can derive **Bayes rule**:

$$P(X = x|Y = y) = \frac{P(Y = y|X = x)P(X = x)}{P(Y = y)}$$

Two random variables X and Y are **independent** if knowing the value of one of the variables does not give us any further clues as to the value of the other variable. Thus, for X and Y independent, $P(X = x|Y = y) = P(X = x)$, or, written another way, $P(X = x, Y = y) = P(X = x)P(Y = y)$.

The **expectation** of a random variable X with respect to the probability distribution $P(X)$ is defined as $E_P[X] = \sum_x P(X = x)x$ or $E_P[X] = \int P(x)x\mathrm{d}x$, depending on whether the random variable is discrete or continuous. The expectation is a weighted average of the values that a random variable can take on. Of course, this only makes sense for random variables which take on numerical values; this would not work in the example from earlier where the two possible values of the "sex" random variable were "male" and "female".

We can also define the **conditional expectation**, the expectation of a random variable with respect to a conditional distribution: $E_{P(X|Y)}[X] = \sum_x P(X = x|Y = y)x$. This is also sometimes written as $E_P[X|Y]$. Lastly, we are not restricted to taking the expectations of random variables only; we can also take the expectation of functions of random variables: $E_P[f(X)] = \sum_x P(X = x)f(x)$. Note that we will often leave the probability distribution implicit and write the expectation simply as $E[X]$.

Expectation is **linear**, which means that the expectation of the sum is the sum of the expectations, i.e. $E[X + Y] = E[X] + E[Y]$, or, more generally, $E[\sum_{i=1}^N X_i] = \sum_{i=1}^N E[X_i]$. For the two-variable case, this can be proven as follows:

$$\begin{aligned}
E[X + Y] &= \sum_{x,y} P(X = x, Y = y)(x + y) \\
&= \sum_{x,y} P(X = x, Y = y)x + \sum_{x,y} P(X = x, Y = y)y \\
&= \sum_x P(X = x)x + \sum_y P(Y = y)y \\
&= E[X] + E[Y]
\end{aligned}$$

The N-variable case follows from this by induction.

For readability, let $\bar{x} = E[X]$. Then we can define the **variance**, $Var[X] = E[(x - \bar{x})^2]$. In words, the variance is the weighted average of the distance from the mean squared. Why is the distance squared? Well, if we take out the square, we get that $Var[X] = E[x - \bar{x}]$, which by linearity of expectation equals $E[x] - E[\bar{x}] = E[X] - \bar{x} = 0$, so we put the square in to keep that from happening. The reason we do not use absolute value instead is that the absolute value function is nondifferentiable. As a result of squaring, the variance penalizes further outliers more.

4 A Machine Learning Example

Consider a set of N **independent and identically distributed** random variables $X_1, \ldots, X_N$. "Independent and identically distributed", which is usually abbreviated as i.i.d., means that the random variables are pairwise independent (i.e. for each i, j such that $i \neq j$, X_i and X_j are independent) and that they are all distributed according to the same probability distribution, which we will call P. Much of the data that we will look at in this class is i.i.d. Since our goal is to automatically infer the probability distribution P that is the best description of our data, it is essential to assume that all of our datapoints were actually generated by the same distribution. One of the implications of i.i.d. is that the joint probability distribution over all of the random variables decomposes as follows: $P(X_1, \ldots, X_N) = \prod_{n=1}^{N} P(X_n)$.

Again, our task is to automatically infer the probability distribution P which best describes our data. But how can we quantify which probability distribution gives the best description? Suppose that our random variables are discrete and have K possible outcomes such that each datapoint X takes on a value in $\{1, \ldots, K\}$. We can describe P with a K-dimensional vector that we will call θ, letting $\theta_k = P(X = k)$ for each k; θ is called the **parameters** of the distribution. It's useful to describe the probabilities in our distribution using a vector, because then we can employ the powerful tools of vector calculus to help us solve our problems.

Now the question of finding the best probability distribution becomes a question of finding the optimal setting of θ. A good idea would be to pick the value of θ which has the highest probability given the data:

$$\begin{aligned}
\theta^* &= \operatorname*{argmax}_{\theta} P(\theta | X_1, \ldots, X_N) \\
&= \operatorname*{argmax}_{\theta} \frac{P(X_1, \ldots, X_N | \theta) P(\theta)}{P(X_1, \ldots, X_N)} \\
&= \operatorname*{argmax}_{\theta} P(X_1, \ldots, X_N | \theta) P(\theta)
\end{aligned}$$

(We get from the first step to the second using Bayes rule, and from the second to the third by observing that the denominator does not depend on θ and is therefore irrelevant to maximizing with respect to θ.) For the moment, let's assume that all settings of θ are equally probable (although this assumption will be challenged in future lectures). Then our problem becomes a matter of finding

$$\theta^* = \operatorname*{argmax}_{\theta} P(X_1, \ldots, X_N | \theta)$$

This method of estimating θ is called **maximum likelihood estimation**, and we will call the optimal setting of the parameters θ_{MLE}. It is a constrained optimization problem that we can solve using the tools of vector calculus, though first we will introduce some more convenient notation. For each k, let $c(k) = \sum_{n=1}^{N} I(X_n = k)$ be the number of datapoints with value k. Here, I is an indicator variable which is 1 when the statement in the parentheses is true, and 0 when it is false.

Using these counts, we can rewrite the probability of our data as follows:

$$\begin{aligned}
P(X_1, \ldots, X_N | \theta) &= \prod_{n=1}^{N} \theta_n \\
&= \prod_{k=1}^{K} \theta_k^{c(k)}
\end{aligned}$$

This switch in notation is very important, and we will do it quite frequently. Here we have grouped our data according to outcome rather than ordering our datapoints sequentially.

Now we can proceed with the optimization. Our goal is to find $\text{argmax}_\theta \prod_{k=1}^{K} P(X_k)^{c(k)}$ such that $\sum_{k=1}^{K} \theta_k = 1$. (We need to add this constraint to assure that whichever θ we get describes a valid probability distribution.) If this were an unconstrained optimization problem, we would solve it by setting the derivative to 0 and then solving for θ. But since this is a constrained optimization problem, we must use a **Lagrange multiplier**.

In general, we might want to solve a constrained optimization problem of the form $\max_{\vec{x}} f(\vec{x})$ such that $g(\vec{x}) = c$. Here, $f(\vec{x})$ is called the **objective function** and $g(\vec{x})$ is called the **constraint**. We form the **Lagrangian**

$$\nabla f(\vec{x}) - \lambda \nabla g(\vec{x}) = 0$$

and then solve for both $\vec{x}$ and λ.

Now we have all the tools required to solve this problem. First, however, we will transform the objective function a bit to make it easier to work with, using the convenient fact that the logarithm is monotonic increasing, and thus does not affect the solution.

$$\begin{aligned}
\max_\theta \prod_{k=1}^{K} P(X_k)^{c(k)} &= \max_\theta \log\Big(\prod_{k=1}^{K} \theta_k^{c(k)}\Big) \\
&= \max_\theta \sum_{k=1}^{K} \log(\theta_k^{c(k)}) \\
&= \max_\theta \sum_{k=1}^{K} c(k) \log(\theta_k)
\end{aligned}$$

We get the gradient of this objective function by, for each θ_k, taking the partial derivative with respect to θ_k:

$$\frac{\partial}{\partial \theta_k}\Big[\sum_{j=1}^{K} c(j) log(\theta_j)\Big] = \frac{c(k)}{\theta_k}$$

(To get this derivative, observe that all of the terms in the sum are constant with respect to θ_k except for the one term containing θ_k; taking the derivative of that term gives the result, and the other terms' derivatives are 0.)

Thus, we get that

$$\nabla f = \frac{\partial}{\partial \theta}\Big[\sum_{j=1}^{K} c(j) log(\theta_j)\Big] = \big(\frac{c(1)}{\theta_1}, \ldots, \frac{c(K)}{\theta_K}\big)$$

In a similar vein,

$$\nabla g = \frac{\partial}{\partial \theta}\Big[\sum_{j=1}^{K} \theta_j\Big] = \big(1, \ldots, 1\big)$$

Now we substitute these results into the Lagrangian $\nabla f - \lambda \nabla g = 0$. Solving this equation, we discover that for each k, $\frac{c(k)}{\theta_k} = -\lambda$, or $\theta_k = -\frac{c(k)}{\lambda}$. To solve for λ, we substitute this back into our constraint, and discover that $\sum_{k=1}^{K} \theta_k = -\frac{1}{\lambda} \sum_{k=1}^{K} c(k)$, and thus $-\lambda = \sum_{k=1}^{K} c(k)$. This is thus our normalization constant.

In retrospect, this formula seems completely obvious. The probability of outcome k is the fraction of times outcome k occurred in our data. The math accords perfectly with our intuitions; why would we ever want to do anything else? The problem is that this formula overfits our data, like the curve separating the male datapoints from the female datapoints at the beginning of class. For instance, suppose we never see outcome k in our data. This formula would have us set $\theta_k = 0$. But we probaby don't want to assume that outcome k will never, ever happen. In the next lecture, we will look at how to avoid this issue.